Leveraging Knowledge Graphs for Enhanced Text Classification of CNN Articles

1st Arwa Bader
Data Science Department
Princess Sumaya University for Technology
Amman, Jordan
arw20228003@std.psut.edu.jo

2nd Rami Khrais
Data Science Department
Princess Sumaya University for Technology
Amman, Jordan
ram20228062@std.psut.edu.jo

3rd Aus Arafat
Data Science Department
Princess Sumaya University for Technology
Amman, Jordan
aus20228071@std.psut.edu.jo

4th Omar Alqawasmeh
Data Science Department
Princess Sumaya University for Technology
Amman, Jordan
o.alqawasmeh@psut.edu.jo

***Abstract*—This paper delves into the exciting possibility of leveraging knowledge graphs (KGs) to empower machine learning models for text classification tasks. We posit that by incorporating the semantic relationships embedded within KGs, we can significantly enhance model performance. To investigate this, we utilize embeddings derived from graph walks conducted over KGs constructed specifically from article features. Three well-established gradient boosting algorithms – XGBoost, LightGBM, and AdaBoost – are employed for two sets of models. The first set acts as a baseline, utilizing the raw text features directly. The second set harnesses the power of KGs by incorporating embeddings derived from the graph walks. The evaluation results are highly promising, demonstrating a clear advantage for models that incorporate knowledge graphs. Across all key metrics – accuracy, recall, precision, and F1-score – the KG-based models consistently surpassed their baseline counterparts. Interestingly, XGBoost exhibited the most significant improvement, suggesting its particular ability to capitalize on the rich information encoded within the graph walk embeddings. Our results shed light on the transformative power of knowledge graphs (KGs). By incorporating KGs, we can enrich text representations, leading to more accurate classifications. This research opens exciting doors for future exploration. We can delve deeper to understand which specific aspects of KGs contribute most significantly to improved performance. Additionally, investigating the effectiveness of different techniques for constructing KGs and their impact on text classification tasks would be valuable. Finally, exploring this approach in various text classification domains, beyond the one we studied, could broaden our understanding and the applicability of this methodology. By continuing to integrate KGs with machine learning for text classification, we can unlock new possibilities. This has the potential to revolutionize how we extract meaning and insights from vast amounts of text information.**

***Index Terms*—Knowledge Graph, Machine Learning, NLP, Text Classification.**

I. Introduction

In the world of text classification, machines are getting smarter with the power of knowledge graphs (KGs). Forget the old way of just reading text one line at a time. KGs are like digital filing cabinets filled with information about things and how they're connected. This lets machines see the bigger picture and understand the relationships between words. It's like giving them a superpower to grasp the true meaning of text, leading to much more accurate classifications. [19]

There are a few key reasons why knowledge graphs (KGs) are becoming a game-changer for text classification. Unlike traditional methods that just analyze words on their own, KGs provide a much richer understanding of the text's meaning. They do this by adding a layer of semantic context, which basically means they show how words relate to each other and the broader world. This is especially helpful for complex texts, where the true meaning can be hidden between the lines. In fact, research by Hogan et al. [7] on knowledge graphs for AI highlights how integrating semantic knowledge can significantly improve the accuracy of text analysis systems. This is because KGs allow the system to understand the relationships between words, not just the words themselves. [7] Moreover, Building on the idea of richer context, Ehrlinger et al. explored how knowledge graphs can be used for semantic data management. This directly translates to more efficient text classification. Knowledge graphs organize information in a way that makes it easier for machines to understand the relationships between words. This efficient organization leads to faster and more accurate text classification. [4]. Adding to the excitement, research by Paulheim et al. delves into the world of combining knowledge graphs (KGs) with powerful neural networks, especially those used in deep learning. Their work shows that these integrated systems can outperform traditional models when tackling complex text classification tasks. Imagine a system that can not only analyze the words themselves but also leverage the rich connections within KGs. This is exactly what Paulheim et al. explore, demonstrating how these combined systems achieve superior results in classifying intricate text data. [18]. This hybrid approach combines the best of both: knowledge graphs and neural networks. Here the knowledge graphs provide a structured way

to organize information. This structure helps train powerful neural networks. These neural networks become even better at picking up an accurate evidence in language, both the words themselves and their deeper meanings. This leads to much more accurate classifications, especially for text data. The synergy between deep learning and knowledge graphs not only enhances model interpretability but also significantly boosts performance across various text classification benchmarks. While NLP techniques can handle the problem of text classification, researchers seek to explore the application of knowledge graphs for this task, highlighting its transformative impact on the way textual data is analyzed and understood. As the field progresses, the challenges of scalability and real-time processing in knowledge graph-integrated systems remain critical research areas. We aim to investigate the usage of knowledge graphs in classifying texts into categories, specifically CNN articles. Our research aims to answer the following questions:

- How can knowledge graphs be effectively integrated with XGBoost, AdaBoost, and LightGBM models for text classification of CNN articles?
- How do knowledge graph embeddings help in text classification?
- What role do random walks on knowledge graphs play in improving the semantic understanding of article content for text classification tasks?

To the best of our knowledge, none has employed this dataset for text classification using knowledge graphs, so our main contributions are as follows:

- Employing CNN articles dataset for text classification and using both traditional embeddings and knowledge graph embeddings.
- Comparing the performance of three different classifiers, namely, XGBoost, AdaBoost, and LightGBM, in classifying articles to their predefined categories.

II. Related Work

A. Machine Learning Approaches

The application of machine learning techniques to the task of text classification are numerous. This section will discuss such research papers and their approaches and results.

In [1] traditional machine learning algorithms such as Naive Bayes(NB), K Nearest Neighbour(KNN) and Racchio Classifiers were used and compared. To classify text into normal or malicious in a web activity log, finding that NB outperformed both.

In [12] they used KNN approach to classify websites, by classifying web-pages. And then extending the classification onto the website itself. Supplementing traditional KNN with feature selection and term weighting. Improving classification accuracy over using the homepage alone by around 30%.

In [20] attempted to include Term frequency – inverse document frequency (TF-IDF) as a weight measure in their KNN implementation. Generating TF-IDF matrices for each document. The results showed that as number of documents and more importantly categories increases. The accuracy of the model decreases.

In [17] the authors implement a a least square support vector machine (LS-SVM) along with latent semantic indexing (LSI) in an attempt to classify document titles. As well as comparing their approach with a NB and KNN approach, outperforming both. With an achieved 99.9% classification accuracy.

In [11] the authors did not approach the problem as a multi-class classification problem. Developing an approach called Hierarchical Deep Learning for Text classification (HDLTex). Which is composed of stacked DL networks that classify the documents with deeper understanding of each topic.

In [14] a Convolutional Neural Network (CNN) approach for Extreme multi-label text classification (XMTC) was attempted. Where documents were tagged with a set of most relevant categories form a set of up to 670,000. The evaluation was done against 7 state-of-the-art methods on 6 benchmark datasets, showing best or second best performance. Showing the power of DL over complex large text classification tasks.

Another implementation of XMTC was done in [5] presenting Hierarchical Label Set Expansion (HLSE). As well as exploring the effect of different Word Embedding techniques on the performance. Showing that the addition of dependency tree and part of speech embedding does not improve performance and in some cases produces worst results.

In [9] authors proposed a hybrid architecture composed of of deep belief network (DBN) and softmax regression. By first training the DBN and softmax regressor independently, then combining them as a whole while fine tuning parameters.

Although most researchers find impressive accuracy using ML approaches. Most papers describe the same issues encountered when scaling up the document count or the category count. Either from a reduced accuracy or an increase in processing time. With the advent of DL and its data driven models, further research was done in the domain of text classification using these techniques. Showing state-of-the-art performance.

In [21] they implemented a CNN with a Concentration mechanism alongside an N-Gram model to extract key words from a text. Attempting to classify short text data, achieving a precision of 92.6%. As well as achieving a higher precision than static CNN, Showing great promise using this feature extraction method.

In [13] researchers explored the implementation of a Bidirectional Long-Short Term Memory along side a CNN (Bi-LSTM-CNN). To encode both forward and backward semantics of text, and applied the model to news text classification. With the model achieving an accuracy of 96.45%, and an F1 score of 0.99. With the proposed architecture outperforming traditional machine learning approaches as well as CNN and LSTM showing great potential.

B. Knowledge Graph Approaches

In this section, we will discuss enhancements of performance of text classification, by using Knowledge Graphs (KG)

to improve the semantics of the embeddings over Machine Learning.

In an attempt to remedy the sparseness of and lack of context in short text classification, [10] use a Background Knowledge Graph along with a graph neural network. To capture both the information retrieved via the triples, and utilize the structural information of the graph itself. The proposed model achieved an accuracy of 80% and performed better than other state of the art models.

While [15] attempted to classify Weibo text by also enhancing the semantics of short text using knowledge graph. The generated embeddings were then fed into a TextCNN, to generate 1 of 10 classes. The results were compared to TextCNN without KG and showed an improvement in F1 score by 11%.

In [22] KG was used to enhance the word embedding of the fast text pre trained model. As well as resolve the Out Of Vocabulary (OOV) problem, by using KG to generate embeddings for such words. The resulting word embeddings were fed into a GRU model for text classification. With improved accuracy over the state-of-the-art across multiple benchmark datasets.

In order to comparison between the whole related works you can see table. I

III. Dataset

Our research utilizes a text classification dataset obtained from Kaggle [6]. This dataset contains around 38,000 news articles gathered from CNN between 2011 and 2022. The data was collected using a web crawler, allowing for potential customization in future iterations [6]. The specific details extracted include author, publication date, category, article section, keywords, second headline, URL source, headline, description, and full text [6]. This comprehensive range of features provides a rich resource for text classification tasks. Notably, the dataset creator suggests leveraging these various elements – category, article section, headline, and description – to create the most informative classification labels [6]. This aligns perfectly with our chosen task of classifying news articles into seven distinct categories: news, sport, politics, business, entertainment, health, and travel. We have excluded the categories "style" and "VR" due to the limited number of samples (1 and 5, respectively), which would hinder the effectiveness of our machine-learning models. The remaining categories, with a distribution of news (18,069), sport (15,541), politics (2,461), business (854), health (557), travel (39), and entertainment (413) articles, provide a more balanced dataset for our text classification task. This balance is crucial for ensuring our models can learn effectively from each category. Overall, the abundance of data points across various news-related categories makes this dataset well-suited for our investigation into effective text classification methods. However, due to time constraints, we filtered out a number of data in each category to use in our task. Table II lists the number of records in each category.

A. Data Pre-Processing

The initial stage of our data pre-processing tackles missing values. We employ a method to meticulously remove any rows containing null values from the dataset. This ensures a clean and consistent foundation for the classification process, eliminating potential errors or inconsistencies that could arise from missing information. Our data initially contained around 38,000 rows. After this cleaning step, we are left with approximately 37,991 rows (9 rows were removed).

Following the null value removal, we focus on standardizing the format within the "Author" column. This step addresses a potential inconsistency where author names might be inconsistently formatted. We implement a process to examine each cell in the "Author" column meticulously. If a value is present, it is split on commas, and only the first element (the author's name) is retained because sometimes the author's name is followed by "CNN," like "Jacopo Prisco, CNN." This ensures a uniform format for author names throughout the dataset, which can be advantageous for subsequent text-processing tasks.

The second phase of pre-processing prepares the text data for sentiment analysis. Here, we transform the raw article text into a format suitable for sentiment polarity calculation. We leverage the Natural Language Toolkit (NLTK) library. The process involves several steps. First, we convert all text to lowercase for consistency in sentiment analysis. Next, the text is tokenized, splitting it into individual words. Stop words, common words like "the" or "is" that don't significantly contribute to sentiment, are removed. Finally, lemmatization is applied, reducing words to their base form (e.g., "running" becomes "run"). This captures the sentiment of words with different grammatical variations.

Following text pre-processing, a sentiment analysis function is implemented using the VADER sentiment analyzer from NLTK, specifically designed for social media contexts and well-suited for analyzing news article text. This function calculates a sentiment score for each article, ranging from -1 (highly negative) to 1 (highly positive), reflecting the overall emotional tone of the article text. By transforming the raw text data through these pre-processing steps, we prepare it for sentiment analysis. The resulting sentiment scores, stored in a new "polarity" column, allow us to explore the emotional leaning of the news articles within our dataset and gain valuable insights for downstream tasks. Using sentiment analysis with textual data can indeed enrich a knowledge graph with semantic information. Sentiment analysis facilitates comprehension of the attitudes, beliefs, and feelings conveyed in textual material. By offering new perspectives on the feelings connected to various entities or connections shown in the graph, this semantic layer can greatly improve a knowledge graph. [16]

Our pre-processing pipeline extends beyond sentiment analysis to incorporate Named Entity Recognition (NER). NER is a critical technique in natural language processing that allows us to identify and categorize specific entities within text data. These entities can encompass people, organizations, locations, dates, monetary values, etc.

TABLE I
COMPARISON OF RELATED WORK APPROACHES FOR TEXT CLASSIFICATION

Feature	Machine Learning Approaches	Knowledge Graph Approaches
Algorithms	Naive Bayes (NB), K Nearest Neighbors (KNN), Support Vector Machines (SVM), Latent Semantic Indexing (LSI)	Background Knowledge Graph, Graph Neural Networks, TextCNN
Focus	Statistical learning from text data	Incorporating semantic relationships from knowledge graphs
Strengths	Efficient for smaller datasets, interpretable results (for some models)	Improved performance on complex and short text data, captures semantic context
Weaknesses	Accuracy can decrease with large datasets or many categories, limited ability to capture complex relationships	Requires building and maintaining knowledge graphs
Examples	Classifying web activity logs (normal vs. malicious)	Classifying short text data (social media posts, news articles)
Performance	Up to 99.9% accuracy reported for specific tasks	Up to 11% improvement in F1-score and 80% accuracy reported
Scalability	Accuracy can decrease with large datasets	May require additional computational resources for knowledge graph processing
Future Directions	Exploring hybrid approaches with deep learning	Investigating different knowledge graph construction techniques and their impact on classification

TABLE II
DATA FILTERED BY CATEGORY

Category	Number of Data Filtered
News	2800
Sport	2800
Politics	2461
Business	854
Health	557
Entertainment	413
Travel	39

In this stage, we leverage a powerful NLP toolkit, spaCy, to employ its pre-trained English NER model. This model is adept at recognizing a broad range of entity types, including commonly encountered categories like people, organizations, and locations. We define a list of these entity types to guide the NER process.

We wrote a special function to carefully examine the text of each article. This function uses the powerful of spaCy library to identify and categorize the important names and details within the text, like people, organizations, and locations. These categories were defined beforehand. The function goes through each identified name or detail and stores them in a well-organized format, grouped by their corresponding category. To make things easier to understand later, the code also combines any entries that fall under the same category, like grouping all the mentions of different people together.

Once we've identified the important details (like people, organizations, and locations) in each news article, we take another step to add even more information to our data set. We created a new function that goes through all the articles again. This function adds new columns to our data table, one for each type of detail we're interested in. At first, these new columns are empty. Then, the function runs the entity extraction process (the one we described earlier) on each article's text. Finally, it fills in the corresponding new columns in the data table with the details it finds. This way, we can easily see how often each type of detail appears in each article.

By pinpointing the important names and details (like people, organizations, and locations) in each news article, we're enriching our data with valuable information. This allows us to ask more interesting questions later on. For example, we could see which types of entities appear most often in different news categories, or even explore how the sentiment of an article might be connected to the people or organizations mentioned. This extra layer of detail helps us understand the news articles in a much richer way. It's like having a more complete picture of what's going on in the stories themselves.

Taking this a step further, we can even connect these named entities to a bigger knowledge graph. This is like a giant web of information where all these details are linked together. By doing this, we can uncover even deeper insights and relationships between different entities. It's like having a more powerful tool to explore and understand the news as a whole. [8] As we aim to do more future research using this dataset.

The last step in preparing our data involves delving deeper into the actual writing of the news articles. We do this by extracting informative features, like little pieces of information, from the text itself. These features help machine learning models, which are powerful tools for analyzing data, to understand the different characteristics of the writing. This, in turn, helps the models classify the articles more accurately.

Our approach uses several techniques to create a well-rounded set of features. First, we look at basic things like the number of words and characters in each article, along with how many sentences there are. We also calculate the average word and sentence length. These features give us a sense of the overall writing style and complexity of the news articles.

To gain a deeper understanding of the writing style in each article, we delve into the grammatical structure. We use a powerful tool called spaCy, which acts like a language expert, dissecting each sentence and identifying the role of every word. By counting the different types of words – nouns, verbs, adjectives, and so on – we can build a profile of the overall writing style. For instance, articles packed with verbs might be more focused on describing actions, while those brimming with adjectives might paint a more vivid picture through detailed descriptions.

We also explore the richness of the vocabulary used. One way we do this is by looking at the ratio of unique words to the total number of words. A higher ratio suggests a more diverse vocabulary, sometimes indicating a more complex writing style. Additionally, we identify words that appear only once in each article. These might be specialized terms or unique phrases that could offer clues about the specific topic or content of the news story.

By incorporating these text features alongside the previously extracted sentiment scores and named entities, we create a richer dataset that captures various aspects of the news article text data. This comprehensive feature set provides a stronger foundation for training and evaluating machine learning models for our text classification task. Moreover, it enriches a knowledge graph by providing detailed attributes about the text that capture writing style, complexity, grammatical structure, and vocabulary richness. [3] [2] [23].

As shown in fig. 1, we describe our data pre-processing pipeline.

IV. Knowledge Graph Construction

At this stage, we started building our knowledge graph, aiming to classify CNN news articles; our primary objective is to enhance the classification of articles into predefined categories. We enriched our knowledge graph with all semantics extracted, as mentioned in the preprocessing section. Our knowledge graph can help encapsulate not just the content of individual news articles but also their interrelationships and contexts. This can lead to a better understanding of how different news events are connected or how certain topics evolve over time. For text classification, this enriched context allows our models to make more informed classifications based on not just the content of a single article but its context within the larger news ecosystem. Further more, By structuring data in a graph format, it becomes easier to navigate and discover related content. This can be particularly useful in text classification; the knowledge graph can help identify closely related categories, aiding in more accurate classification, especially for articles that may span multiple topics.

The process of building our knowledge graph went as follows:

- We created an ontology named "Article Ontology" to represent the structure of CNN articles dataset and the relationships between various entities and topics.
- We defined the terminological box (Tbox) by defining the main classes and properties; for instance, our ontology includes the primary classes: Article Text, Author, Category, and Section. The extracted features were related to the Article Text class as a data property with an appropriate predicate; 2 shows how features related to the article text entity.
- We then extracted all triples from the dataset, and by employing the classes and predicates defined in the Tbox, this is done to build an assertion box (Abox) compatible with the Tbox.
- After that, we Built the knowledge graph; we utilized Python to facilitate the integration of ABox data into the TBox, ensuring a seamless mapping between instance-specific data and the overarching ontological schema defined in the TBox. Python's libraries and frameworks enabled efficient handling of textual data and interaction with the ontology.

We employed GraphDB to store and visualize the knowledge graph we constructed. As illustrated in Figure 3, a snippet from the knowledge graph is displayed, showcasing an article node. This particular node is classified under the 'Article' class, tagged with the 'news' category, and assigned to the 'Europe' section. Additionally, the article is connected with its author, John Allen. This example underlines just a few numbers of the nodes connected to this article, indicating the rich semantic relationships that our knowledge graph establishes for every article. With the help of this semantic enrichment, each piece of content can be fully understood in terms of the relationships and characteristics that are connected to it.

Figure 4 illustrates some articles belonging to the category News; it's worth noting that this is a small number of articles since our knowledge graph contains 10K articles belonging to this category.

V. Methodology

A. Machine Learning Models (baseline)

To categorize the news articles into different classes, we initially opted for a machine-learning approach. We employed three well-established algorithms: XGBoost, AdaBoost, and LightGBM. These algorithms are known for their effectiveness in various classification tasks. However, before feeding the text data directly to the models, we incorporated an additional step of Word2Vec. This technique transforms each word within the article into a numerical representation, essentially capturing the semantic meaning and relationships between words. By doing this, we hoped the machine learning models could better understand the nuances of the text and achieve a more accurate classification.

Following the machine learning approach, we wanted to explore the potential of knowledge graphs. These knowledge graphs act as vast information networks where entities (like people, organizations, and locations) and their relationships are explicitly linked. We hypothesized that by leveraging the rich information within the knowledge graphs alongside the text features, we could achieve even better classification results. The following sections will detail the development of our knowledge graph-based approach and compare its performance against the initial machine learning baseline. This comparison will help us determine the most effective strategy for classifying the news articles within our dataset.

B. Knowledge Graph Embedding

We constructed node embeddings from a knowledge graph derived from Turtle (TTL) format RDF data. The process integrates the use of RDFlib for RDF graph manipulations, Net-

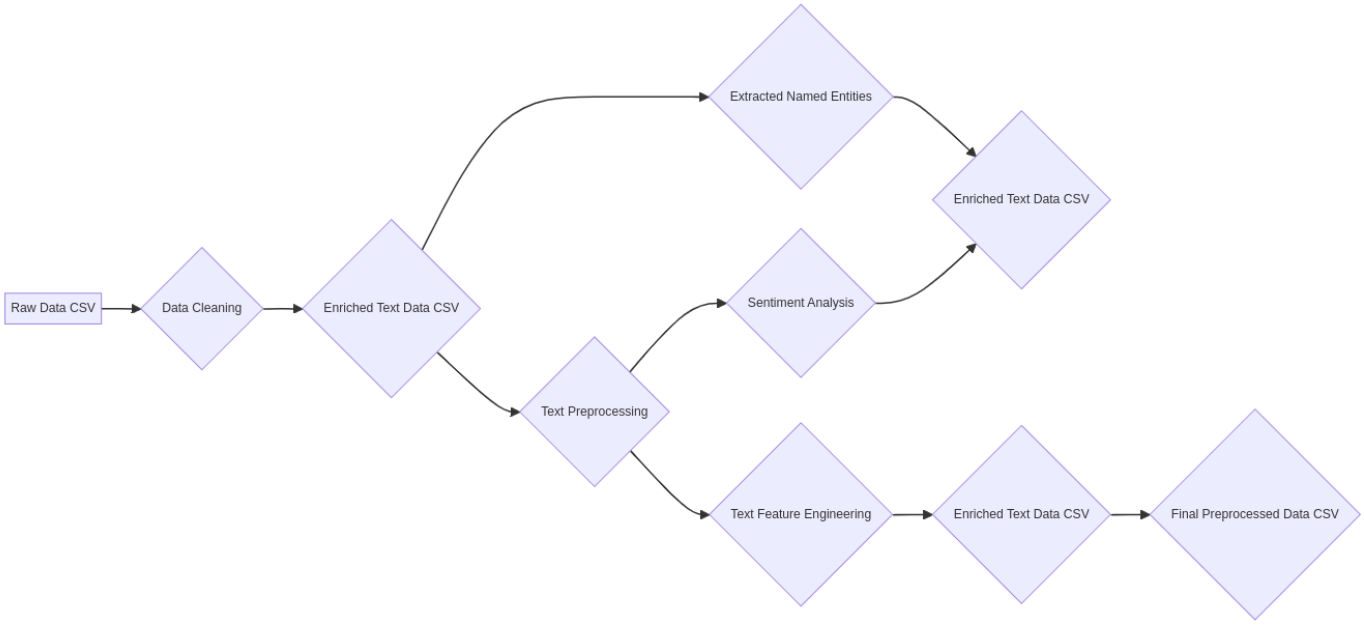

Fig. 1. The data pre-processing pipeline.

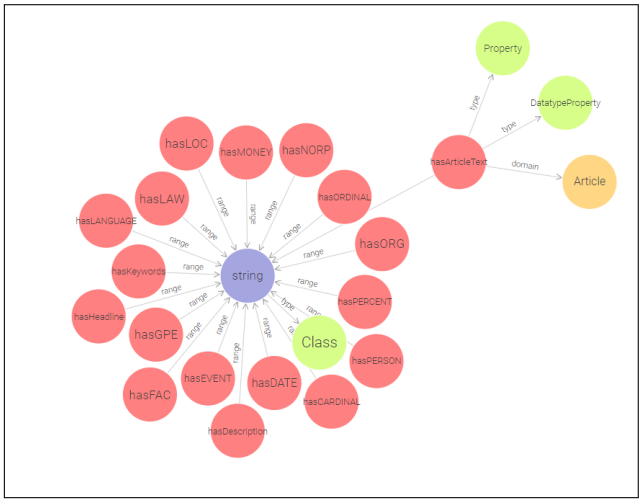

Fig. 2. Article text and its related features.

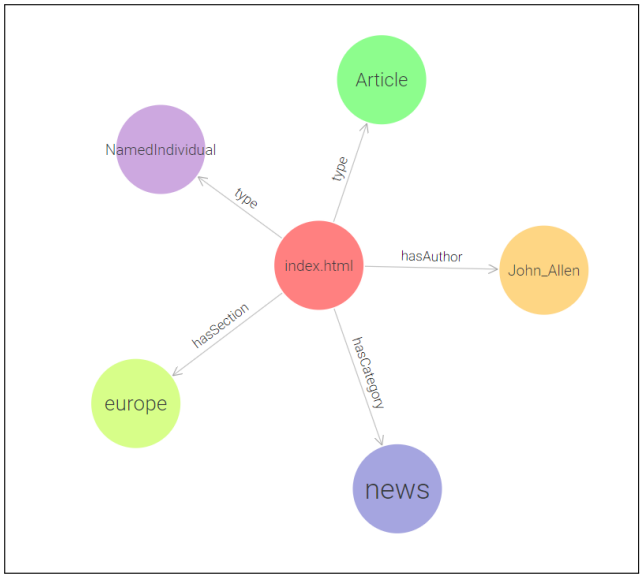

Fig. 3. An article knowledge graph

workX for graph-based operations, Node2Vec for generating random walks, and Word2Vec for learning node embeddings. Below, we detail each step of the procedure.

1) Graph Construction from RDF Data: We started with RDF graph initialization; utilizing RDFlib, a prominent Python library for RDF operations, we initiated an empty RDF graph. This graph serves as a container for the RDF data parsed from the Turtle format file. After that we Parsed the TTL data, where the Turtle file is parsed to populate the RDF graph. Turtle, a text-based format for RDF, offers a compact and readable syntax for expressing RDF triples, which consist of subjects, predicates, and objects. Subsequent to RDF parsing, we instantiate an undirected graph using NetworkX, a powerful library designed for handling complex networks. This graph is constructed by iterating over each RDF triple, where subjects and objects are treated as nodes, connected by edges labeled with RDF predicates. This conversion lays the groundwork for the embedding process by structuring the RDF data in a graph format conducive to network analysis.

2) Feature Extraction via Node2Vec: Extraction of the features starts with selecting a walking strategy that serves the task; we here chose to take random walks by using Node2Vec, an algorithm adapted from Word2Vec specifically for graphs, to generate random walks across the NetworkX graph. These walks are parameterized by:

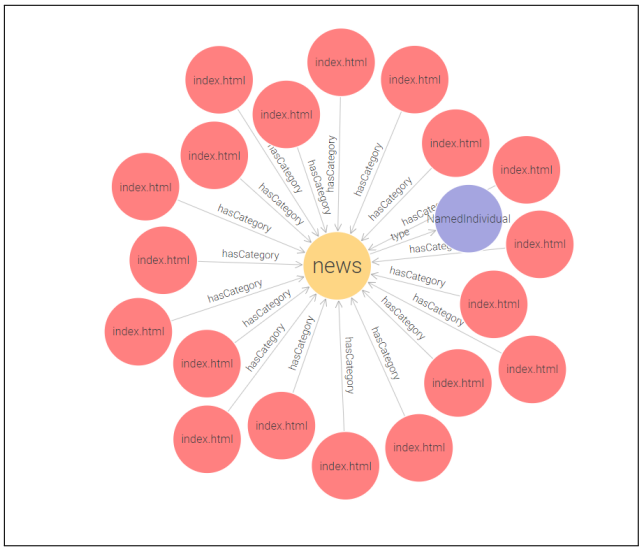

Fig. 4. Articles belonging to news category

- Number of Walks: Dictates how many walks to initiate from each node, enhancing the robustness of the sampling process.
- Walk Length: Determines the number of steps to be taken in each walk, affecting the scope of context captured around each node.
- Return Parameter (p): Influences the likelihood of revisiting a node, thereby adjusting the exploration-exploitation balance.
- In-out Parameter (q): Modulates the walk's focus between exploring outward to new nodes and inward to explore local neighborhoods.

In our case, the number of walks was 100, which in turn helps capture a diverse set of paths and contextual relationships among the nodes, which can then be used to train the embedding model. The walk length was 10 since our graph is dense with semantics. Also, since the parameters p and q play crucial roles in guiding the walk's behavior, specifically how it explores the graph, we sat them to 1 both; for P=1, the walk is equally likely to return as to continue moving away, providing a neutral setting without bias towards returning. With q = 1, the walk is equally inclined to explore local neighborhoods and distant parts of the graph, maintaining a balanced exploration strategy. Using the above parameters, we utilized Walk generation; Node2Vec simulates walks that effectively sample the graph's structure. These walks represent sequences of nodes that capture both the local and global architectural features of the graph, analogous to sentences in natural language processed by Word2Vec.

3) Embedding Learning with Word2Vec: At this stage, we began model training, where the sequences of nodes generated from Node2Vec are used to train a Word2Vec model. This model learns low-dimensional, dense embeddings for each node, where the embedding space captures the structural and contextual nuances discovered through the random walks. Key parameters include:

- Vector Size: Defines the dimensionality of the embeddings, balancing detail with computational efficiency.
- Window Size: Sets the maximum distance between the current and predicted node in a sequence, influencing the local context scope.
- Skip-gram Architecture (sg): Specifies the training algorithm, with skip-gram chosen for its efficacy in learning high-quality embeddings for nodes appearing in diverse contexts.

In our case, the vector size was set to 64, specifying the dimensionality of the feature vectors in which the graph nodes will be embedded. Essentially, this means that each node in the graph will be represented as a vector of 64 continuous values. Moreover, we sat a window of size 5, which fixes the maximum distance between the current and predicted node in a walk. The fixed to sg=1 configuring the Word2Vec model to use the skip-gram algorithm, focusing on learning high-quality embeddings by predicting the context nodes from each target node in the random walks generated from the graph. Also, The negative is set to 10, which is the number of "noise words" to be drawn for negative sampling.

Post-training, the Word2Vec model encapsulates embeddings that abstractly represent each node's role and connections within the graph. These embeddings will be utilized for our classification task, where they will be used as features; the vector representations serve as input features for our classification algorithms to categorize articles into predefined categories based on their content and context within the graph.

C. Machine Leaning Over Knowledge graph

In this section, we delve into the application of machine learning models specifically designed for working with gradient-boosted decision trees for the task of text classification. Here, we leverage the power of knowledge graphs (KGs) by utilizing previously generated embeddings derived from graph walks over KGs constructed from article features. This approach aims to incorporate the inherent semantic relationships within the data into the classification process, potentially leading to more accurate predictions.

In our quest to tackle this text classification challenge, we'll be deploying a powerful machine learning models: XGBoost, LightGBM, and AdaBoost.

XGBoost stands out for its ability to wrangle complex data structures like our graph walk embeddings, all while maintaining efficiency and scalability. Think of it as a juggler who can handle a multitude of information without dropping a beat. This is especially valuable because our embeddings might contain a high number of dimensions, and XGBoost's powerful regularization techniques act like a safety net, preventing the model from overfitting on this complex data.

LightGBM brings a different kind of strength to the table: speed and memory efficiency. When dealing with large datasets containing intricate embeddings, this makes it a serious contender. Imagine being able to analyze a massive

amount of information quickly and efficiently, that's the advantage LightGBM offers.

While AdaBoost might not be the most popular choice for text classification compared to the other two, it plays a crucial role as a baseline for comparison. It works by iteratively refining its predictions, focusing on instances it previously misclassified. This iterative approach can lead to improved overall performance, allowing us to compare the effectiveness of the other models.

We'll train each of these models using the generated graph walk embeddings. The ultimate goal is to have them learn to classify the corresponding text data into their correct categories. To ensure our evaluation is reliable, we'll split our dataset into separate training and testing sets following standard practices.

Once the models have finished training, we'll put them to the test. We'll use well-established metrics for text classification, like accuracy, precision, recall, and F1-score, to evaluate their performance. This will allow us to see how much better the models that use the knowledge graphs (through the walk-based embeddings) perform compared to a baseline model trained only on the original text features. By analyzing how each model does, we can gain valuable insights. We can see which gradient boosting algorithms are best at using the rich information stored in the graph embeddings for text classification tasks. Ultimately, this comparison will reveal whether using the semantic relationships within the knowledge graphs actually leads to more accurate text classification compared to the baseline model.

VI. Result

In this section, we'll dive into the results and evaluation of the machine learning models we used for text classification. We ran two experiments: one where we fed the models the original text features (these are called baseline models), and another where we used the special graph walk embeddings we created from knowledge graphs (these are called KG-based models). For both experiments, we split our dataset 80/20, with 80% used for training the models and 20% used for testing. This ensures a reliable evaluation of how well each model performs.

The baseline models, consisting of XGBoost, LightGBM, and AdaBoost, reached a well results. XGBoost pointed as the strongest performer, achieving an accuracy of 87.6%, a recall of 85.5%, a precision of 88.3%, and an F1-score of 86.9%. These metrics indicate that XGBoost effectively learned the classification patterns within the text data. LightGBM followed closely with an accuracy of 85.1%, demonstrating its capability of handling the complexities of text classification tasks. AdaBoost, while achieving a lower accuracy of 80.9%, still provided a valuable baseline for comparison.

The second experiment, utilizing the KG-based models, yielded even more encouraging results. XGBoost again displayed exceptional performance, achieving a significant improvement over the baseline with an accuracy of 90.7%, a recall of 91.3%, a precision of 90.5%, and an F1-score of 90.9%. This suggests that incorporating semantic relationships captured by the knowledge graphs through the walk-based embeddings significantly enhanced the model's ability to classify the text data accurately. LightGBM also benefitted from the KG information, achieving an accuracy of 87.0%, a respectable improvement over the baseline. Interestingly, AdaBoost showed the most substantial improvement among the three models, reaching an accuracy of 86.1%. This suggests that the knowledge graph information might have been particularly beneficial for addressing the challenges faced by AdaBoost in the baseline experiment. a comparison of the machine learning models are listed in table. III

The results are clear: using knowledge graphs (KGs) significantly boosts the accuracy of text classification. By adding the semantic relationships within the data into the mix, the KG-based models consistently outperformed the baseline models across all the metrics we used to measure performance. This finding highlights the exciting potential of knowledge graphs! They enrich text representations, leading to more accurate classifications. Interestingly, the improvements observed in XGBoost and AdaBoost were particularly impressive. This suggests that these models are especially adept at extracting and utilizing the valuable information encoded within the graph walk embeddings. Future research could delve deeper and explore which specific aspects of the knowledge graphs contribute most significantly to these improvements.

TABLE III
Performance Comparison of our Machine Learning Models.

Model	Accuracy	Recall	Precision	F1-Score
Baseline Models				
XGBoost	87.6%	85.5%	88.3%	**86.9%**
LightGBM	85.1%	83.3%	85.1%	84.2%
AdaBoost	80.9%	78.0%	82.1%	80.0%
KG-based Models				
XGBoost	90.7%	91.3%	90.5%	**90.9%**
LightGBM	87.0%	87.7%	86.9%	87.3%
AdaBoost	86.1%	87.5%	84.2%	85.8%

VII. Conclusion

This research explored whether knowledge graphs (KGs) could give machine learning models a boost in classifying text data. We tested three powerful algorithms – XGBoost, LightGBM, and AdaBoost – on two sets of models. The first set, used the raw text features directly. The second set based on the power of KGs by incorporating special codes derived from exploring connections within these knowledge graphs. These connections were built based on the features of the articles themselves.

Models that incorporated knowledge graphs (KGs) consistently beat out the baseline models across the board. This was true for all the metrics we used to measure performance, like accuracy, how well they remembered relevant information, and how precise their classifications were. Interestingly, XGBoost showed the most improvement, suggesting it's particularly good at using the wealth of information hidden within the

graph walk embeddings. These findings highlight the game-changing potential of KGs. They can enrich how text is understood by the models, leading to more accurate classifications. This opens doors for exciting future research. We can delve deeper into understanding exactly what parts of the KGs are most helpful, and explore different ways of building them. We could also try applying this approach to other text classification tasks beyond the one we studied.

There's a lot more to explore. We can dig deeper to understand exactly which parts of the knowledge graphs (KGs) are most helpful for improving performance. It would also be interesting to see how different methods for navigating these knowledge graphs (like taking different paths) affect the results. Additionally, researchers could experiment with various techniques for building KGs in the first place, and see how that impacts text classification tasks. Finally, imagine applying this approach to entirely different areas of text classification, beyond what we studied here. This could broaden our understanding of how well this method works and open doors to even more applications. By continuing to combine KGs with machine learning for text classification, we can unlock new ways to achieve even more accurate results and gain a deeper understanding of the vast amount of text information available. This has the potential to completely change how we extract meaning and insights from text data.

REFERENCES

[1] Juan Jose Garcia Adeva and Juan Manuel Pikatza Atxa. Intrusion detection in web applications using text mining. *Engineering Applications of Artificial Intelligence*, 20(4):555–566, 2007.

[2] Yves Bestgen. Measuring lexical diversity in texts: The twofold length problem. *Language Learning*, 2023.

[3] Alebachew Chiche and Betselot Yitagesu. Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*, 9(1):10, 2022.

[4] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48(1-4):2, 2016.

[5] Francesco Gargiulo, Stefano Silvestri, Mario Ciampi, and Giuseppe De Pietro. Deep neural network for hierarchical extreme multi-label text classification. *Applied Soft Computing*, 79:125–138, 2019.

[6] hadasu92. Cnn news articles from 2011 to 2022, 2022.

[7] Aidan Hogan. Knowledge graphs: A guided tour. In *International Research School in Artificial Intelligence in Bergen (AIB 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[8] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.

[9] Mingyang Jiang, Yanchun Liang, Xiaoyue Feng, Xiaojing Fan, Zhili Pei, Yu Xue, and Renchu Guan. Text classification based on deep belief network and softmax regression. *Neural Computing and Applications*, 29:61–70, 2018.

[10] Xuhui Jiang, Yinghan Shen, Yuanzhuo Wang, Xiaolong Jin, and Xueqi Cheng. Bakgrastec: A background knowledge graph based method for short text classification. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 360–366. IEEE, 2020.

[11] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017.

[12] Oh-Woog Kwon and Jong-Hyeok Lee. Text categorization based on k-nearest neighbor approach for web site classification. *Information Processing & Management*, 39(1):25–44, 2003.

[13] Chenbin Li, Guohua Zhan, and Zhihua Li. News text classification based on improved bi-lstm-cnn. In *2018 9th International conference on information technology in medicine and education (ITME)*, pages 890–893. IEEE, 2018.

[14] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 115–124, 2017.

[15] Yingjun Liu. Weibo text classification based on knowledge graph. In *Journal of Physics: Conference Series*, volume 1827, page 012123. IOP Publishing, 2021.

[16] Fernando Andres Lovera, Yudith Coromoto Cardinale, and Masun Nabhan Homsi. Sentiment analysis in twitter based on knowledge graph and deep learning classification. *Electronics*, 10(22):2739, 2021.

[17] Vikramjit Mitra, Chia-Jiu Wang, and Satarupa Banerjee. Text classification: A least square support vector machine approach. *Applied soft computing*, 7(3):908–914, 2007.

[18] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

[19] Niloofer Shanavas, Hui Wang, Zhiwei Lin, and Glenn Hawe. Knowledge-driven graph similarity for text classification. *International Journal of Machine Learning and Cybernetics*, 12(4):1067–1081, 2021.

[20] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364, 2014.

[21] Haitao Wang, Jie He, Xiaohong Zhang, and Shufen Liu. A short text classification method based on n-gram and cnn. *Chinese Journal of Electronics*, 29(2):248–254, 2020.

[22] Hongzhong Wang, Kun Guo, and Zhanghui Liu. Mixed word embedding method based on knowledge graph augment for text classification. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1618–1623. IEEE, 2019.

[23] Kelly Woods, Brett Hashimoto, and Earl K Brown. A multi-measure approach for lexical diversity in writing assessments: Considerations in measurement and timing. *Assessing Writing*, 55:100688, 2023.